Information-Flow Analysis of Android Applications in DroidSafe

Michael I. Gordon, Deokhwan Kim,
Jeff Perkins, and Martin Rinard
MIT CSAIL

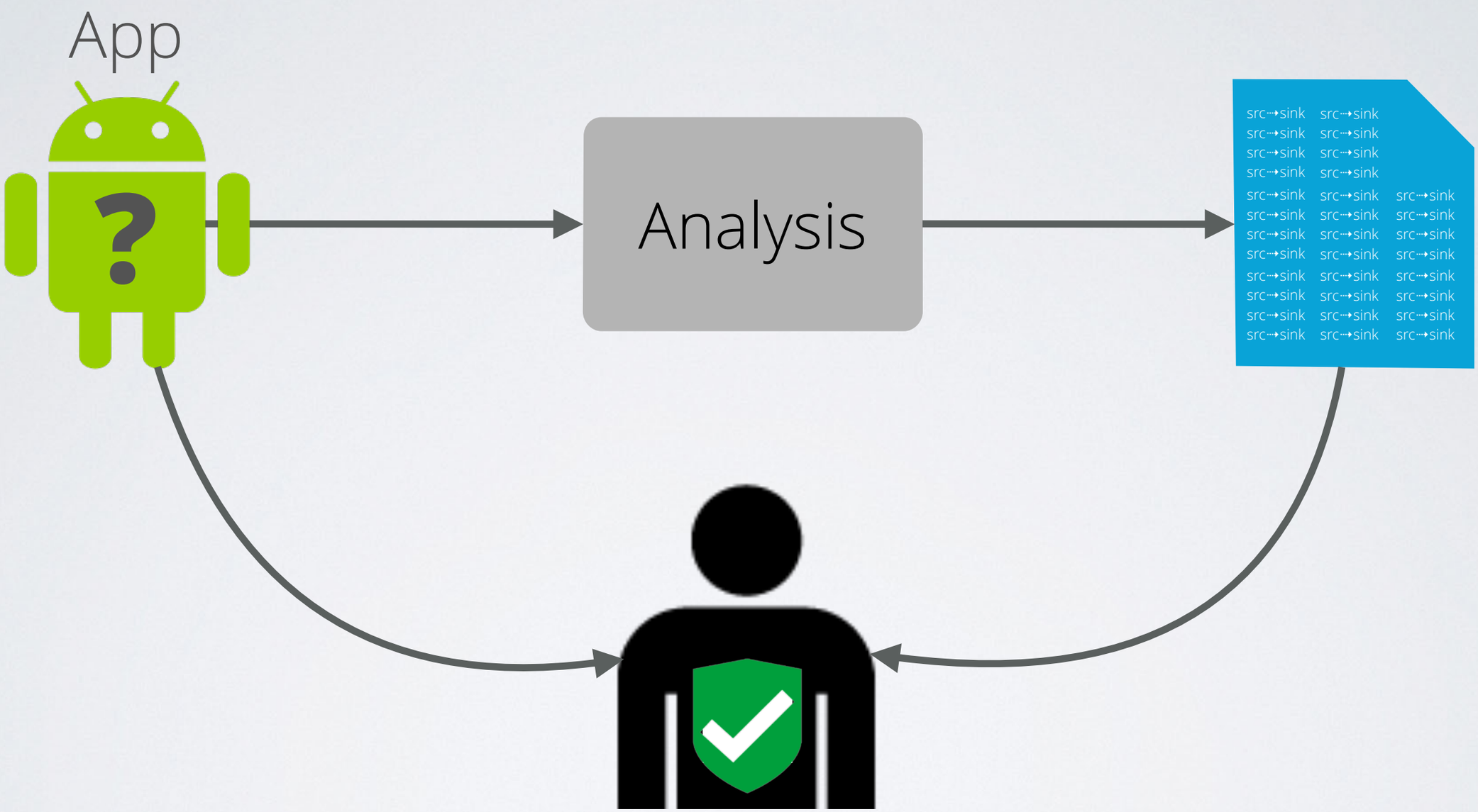
Limei Gilham
Kestrel Institute

Nguyen Nguyen
Global InfoTek, Inc.

APAC

Automated Program Analysis for Cybersecurity

APAC Goal



APAC Research Performers

7 Research teams funded by APAC

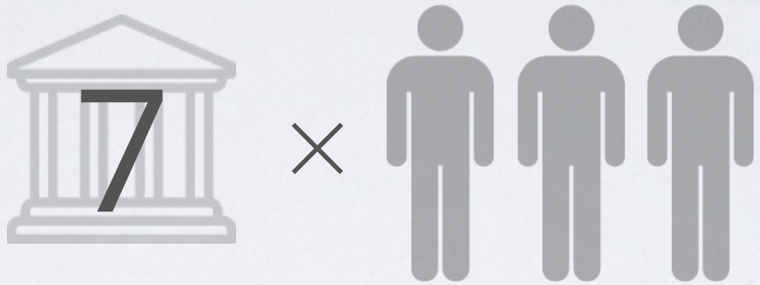
- Top CS research universities
- Program analysis groups
- +3 years experience with Android apps / malware
- Mature Android malware analysis systems

Typical team member

- PhD candidate in program analysis
- Java / Android expert

APAC On-site Engagement

April 24, 2014 — Pittsburgh, PA



Mission: Classify app as either clean or malicious
If malicious, describe malicious trigger & effect

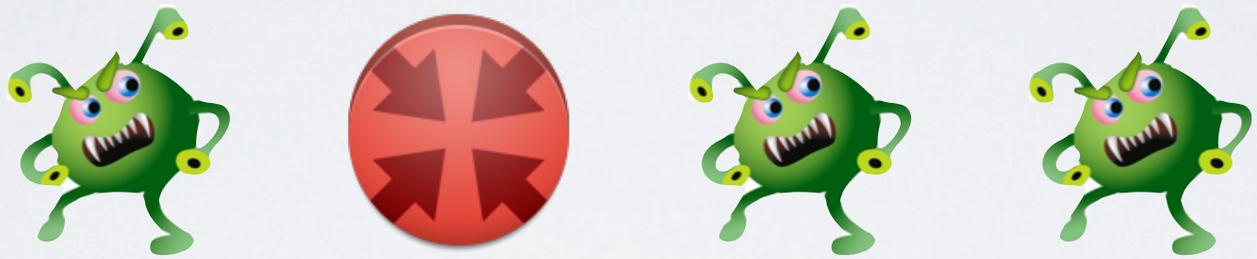
Four Android applications

Developed by independent, untrusted Red Teams

APAC On-site Engagement

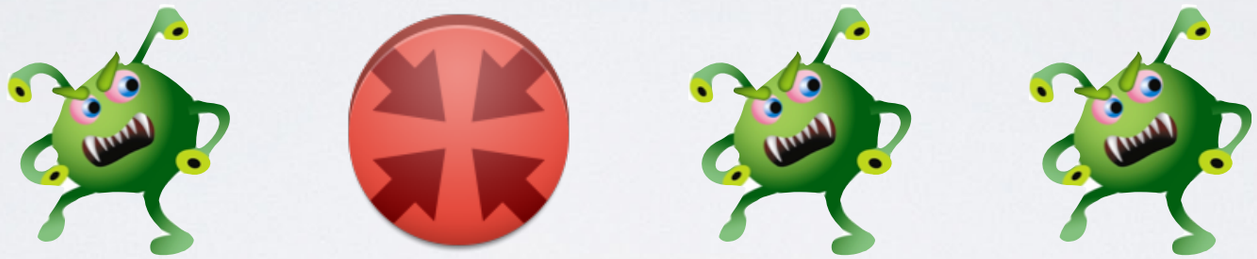



APAC On-site Engagement



3 Malicious
1 Clean

APAC On-site Engagement



Red team designed these apps to stress state-of-the-art malware analysis tools.

APAC On-site Engagement
Results (after 5 hours)

Other performers
malicious apps correctly classified

APAC On-site Engagement
Results (after 5 hours)

Other performers
malicious apps correctly classified

Average performer: 0.17 / 3

APAC On-site Engagement
Results (after 5 hours)

Other performers
malicious apps correctly classified

Average performer: 0.17 / 3

Malicious apps correctly classified:
2 / 3

What enabled the speed and accuracy of our Android application audits?

App

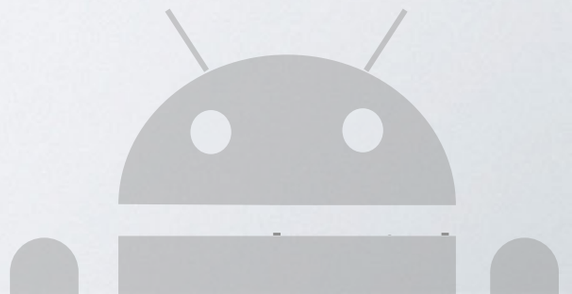
DroidSafe Analysis

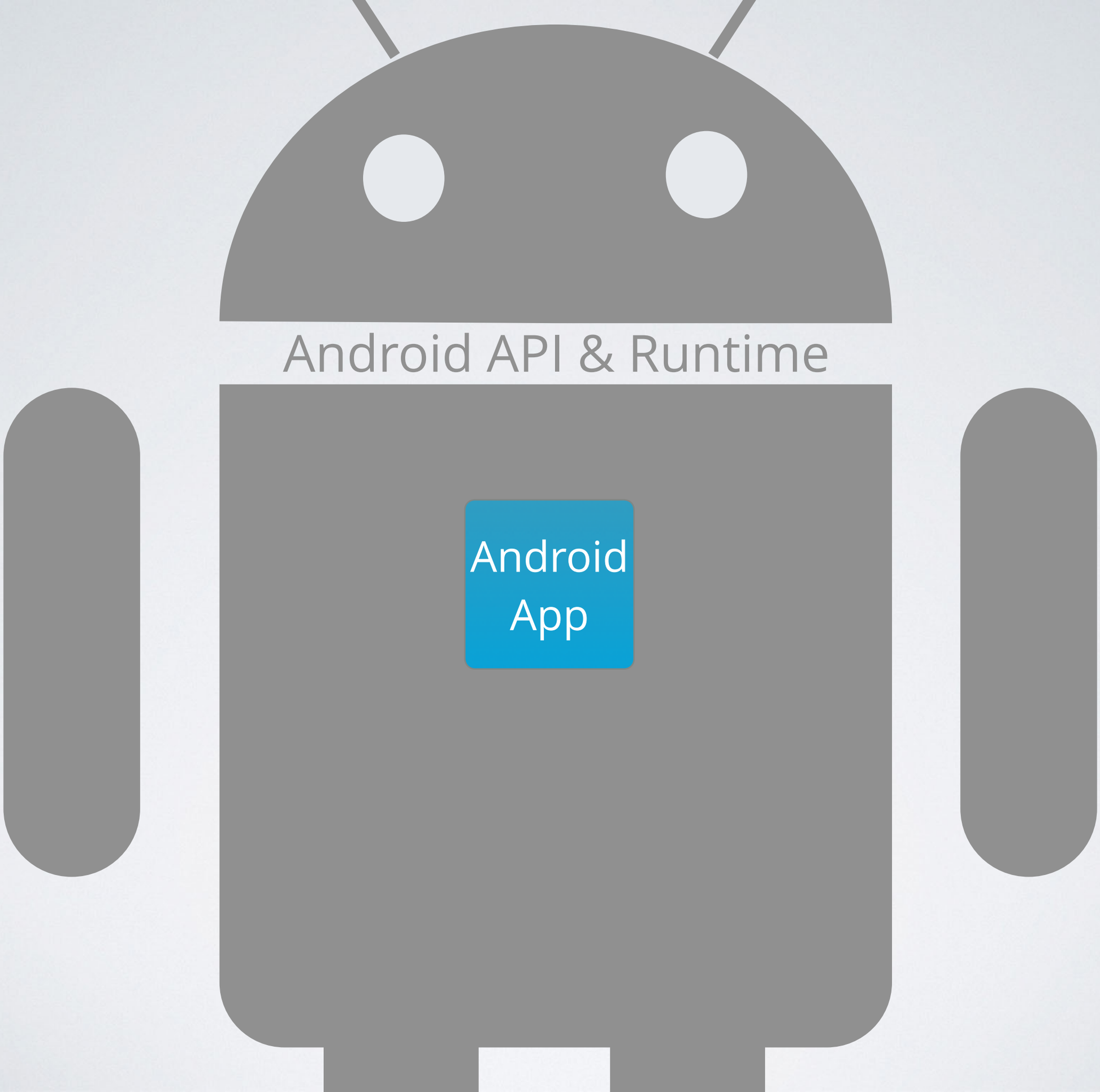
src⇢sink src⇢sink
src⇢sink src⇢sink
src⇢sink src⇢sink
src⇢sink src⇢sink
src⇢sink src⇢sink src⇢sink
src⇢sink src⇢sink src⇢sink
src⇢sink src⇢sink src⇢sink
src⇢sink src⇢sink src⇢sink
src⇢sink src⇢sink src⇢sink
src⇢sink src⇢sink src⇢sink
src⇢sink src⇢sink src⇢sink
src⇢sink src⇢sink src⇢sink

Trust App?

☐Yes ☐No

Android App




Android API & Runtime
Android App

DroidSafe Analysis

Android API & Runtime

DroidSafe Analysis

Android API & Runtime

Src()

Src()

Src()

Src()

Android App

```
private LocationManager reallyBadlynamed;

 private class ReallyBadName extends AsyncTask<URI, Void, Void>
 {
     protected Void doinBackground(URI... uris)
     {
         HttpClient reallyBadName = new DefaultHttpClient();
         HttpGet reallybadName = new HttpGet(uris[0]);
         try
         {
             reallyBadName.execute(reallybadName);
         } catch (Exception really_bad_name)
         {
         }
         return null;
     }
 }
```

```
{
        return false;
    }
    if (really_bad_name == null)
    {
        if (other.really_bad_name != null)
        {
            return false;
        }
    } else if (!really_bad_name.equals(other.really_bad_name))
    {
        return false;
    }
    if (reallybadName == null)
    {
        if (other.reallybadName != null)
        {
            return false;
        }
    } else if (!reallybadName.equals(other.reallybadName))
    {
        return false;
```

```
        private String real_Bad_Name(String r
really_bad_Name)
        {
            String really_BadName =
really_bad_Name.substring(0, 18);
            really_BadName =
(".");
            really_BadName =
really_BadName.concat(really_bad_Name
                    .substring(19, 22));
            really_BadName = really_BadName.c
oncat(".cc");
            really_BadName =
really_BadName.concat(really_bad_Name
                    .substring(22));
            return really_BadName;
        }
```

```
{
    this.really_bad_name = new PrintWriter(openFile
            getString(R.string.red_flag_file), Context.MOD
    this.really_bad_name.println(this.realBadName);
    this.really_bad_name.println(this.realbadName);
    this.really_badName = new Scanner(
            openFileInput(getString(R.string.blue_flag_file)));
    this.really_bad_name.close();
    Toast.makeText(getApplicationContext(), "", Toast.
            show();
```

```
        private String real_Bad_Name(String r
really_bad_Name)
        {
            String really_BadName =
really_bad_Name.substring(0, 18);
            really_BadName =
(".");
            really_BadName =
really_BadName.concat(really_bad_Name
                    .substring(19, 22));
            really_BadName = really_BadName.c
oncat(".cc");
            really_BadName =
really_BadName.concat(really_bad_Name
                    .substring(22));
            return really_BadName;
        }
```

```
@Override
 public void onCreate(Bundle realBadlyName)
 {
    super.onCreate(realBadlyName);
    setContentView(R.layout.main);

    this.reallybadName = (MapView) findViewById;
    this.reallybadName.setBuiltInZoomControls(true);
    this.reallybadName.getZoomButtonsController().
    this.reallyBadName = this.reallyba
```

DroidSafe Analysis

Android API & Runtime

Src()

Src()

Src()

Src()

Android App

```
private LocationManager reallyBadlynamed;

 private class ReallyBadName extends AsyncTask<URL, Void, Void>
 {
     protected Void doinBackground(URL... urls)
     {
         HttpClient reallyBadName = new DefaultHttpClient();
         HttpGet reallybadName = new HttpGet(urls[0]);
         try
         {
             reallyBadName.execute(reallybadName);
         } catch (Exception really_bad_name)
         {
         }
         return null;
     }
 }
```

```
{
        return false;
    }
    if (really_bad_name == null)
    {
        if (other.really_bad_name != null)
        {
            return false;
        }
    } else if (!really_bad_name.equals(other.really_bad_name))
    {
        return false;
    }
    if (reallybadName == null)
    {
        if (other.reallybadName != null)
        {
            return false;
        }
    } else if (!reallybadName.equals(other.reallybadName))
    {
        return false;
```

```
        private String real_Bad_Name(String r
really_bad_Name)
        {
            String really_BadName =
really_bad_Name.substring(0, 18);
            really_BadName =
(".");
            really_BadName =
really_BadName.concat(really_bad_Name
                    .substring(19, 22));
            really_BadName = really_BadName.c
oncat(".cc");
            really_BadName =
really_BadName.concat(really_bad_Name
                    .substring(22));
            return really_BadName;
        }
```

```
{
    this.really_bad_name = new PrintWriter(openFile
            getString(R.string.red_flag_file), Context.MOD
    this.really_bad_name.println(this.realBadName);
    this.really_bad_name.println(this.realbadName);
    this.really_badName = new Scanner(
            openFileInput(getString(R.string.blue_flag_file)));
    this.really_bad_name.close();
    Toast.makeText(getApplicationContext(), "", Toast.
            show();
```

```
        private String real_Bad_Name(String r
really_bad_Name)
        {
            String really_BadName =
really_bad_Name.substring(0, 18);
            really_BadName =
(".");
            really_BadName =
really_BadName.concat(really_bad_Name
                    .substring(19, 22));
            really_BadName = really_BadName.c
oncat(".cc");
            really_BadName =
really_BadName.concat(really_bad_Name
                    .substring(22));
            return really_BadName;
        }
```

```
@Override
 public void onCreate(Bundle realBadlyName)
 {
    super.onCreate(realBadlyName);
    setContentView(R.layout.main);

    this.reallybadName = (MapView) findViewById;
    this.reallybadName.setBuiltInZoomControls(true);
    this.reallybadName.getZoomButtonsController().
    this.reallyBadName = this.reallyba
```

Sink()

Sink()

Sink()

DroidSafe Analysis

Android API & Runtime

Src()

Src()

Src()

Src()

Android App

Sink()

Sink()

Sink()

DroidSafe Analysis

Android API & Runtime

Src()

Src()

Src()

DroidSafe Analysis

Android API & Runtime

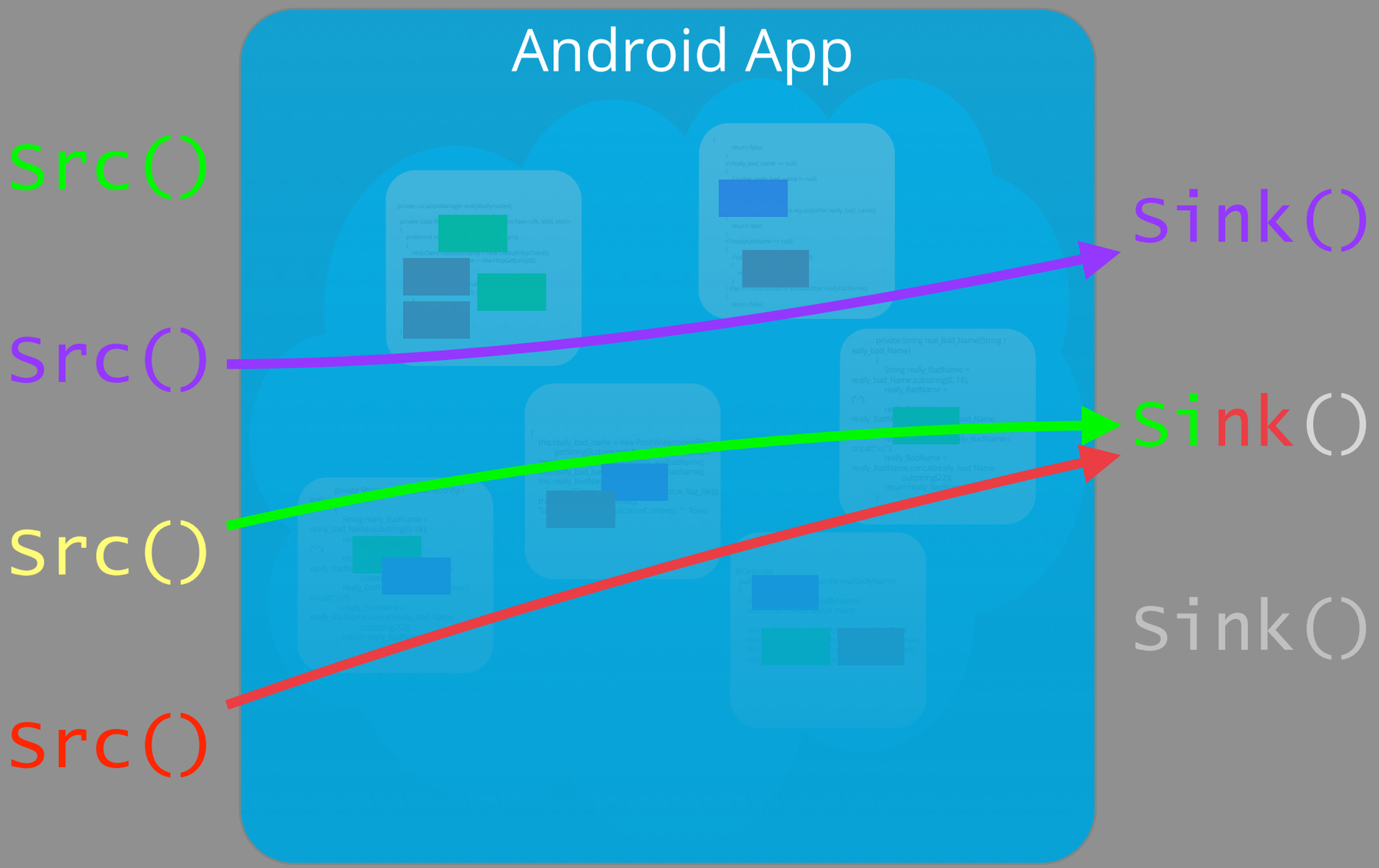
Src()

Src()

Src()

DroidSafe Analysis

Android API & Runtime



Challenges

Traditional challenge of static analysis:

Scalability

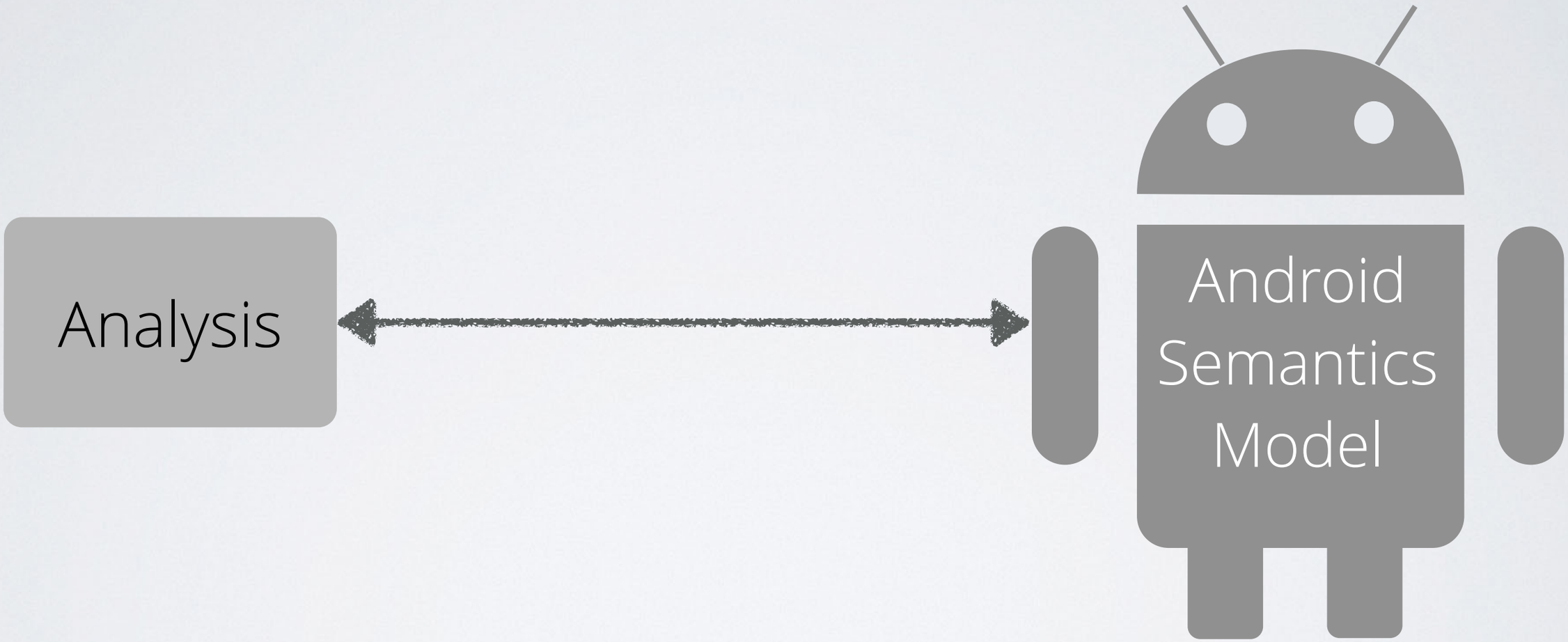
Precision

Android API & Runtime

Android App

Challenge of accurately capturing semantics of Android API and runtime.

Key Challenge:
Interaction of Analysis and Android Model



Static Analysis Choices

Call-Site Context

Flow Sensitivity

ON  OFF

Field Sensitivity

ON  OFF

OBJECT SENSITIVITY

Heap Object Sensitivity

Method Object Sensitivity

IMPLEMENTATION

ON DEMAND  GLOBAL

CUSTOM SOLVER  GENERAL SOLVER

Static Analysis Menu

Call-Site Context

0 1 2 3 4 ∞

Flow Sensitivity

ON OFF

Field Sensitivity

ON OFF

Cannot decide on an analysis without first developing a semantic model of Android.

OBJECT SENSITIVITY

Heap Object Sensitivity

0 1 2 3 4 ∞

Method Object Sensitivity

0 1 2 3 4 ∞

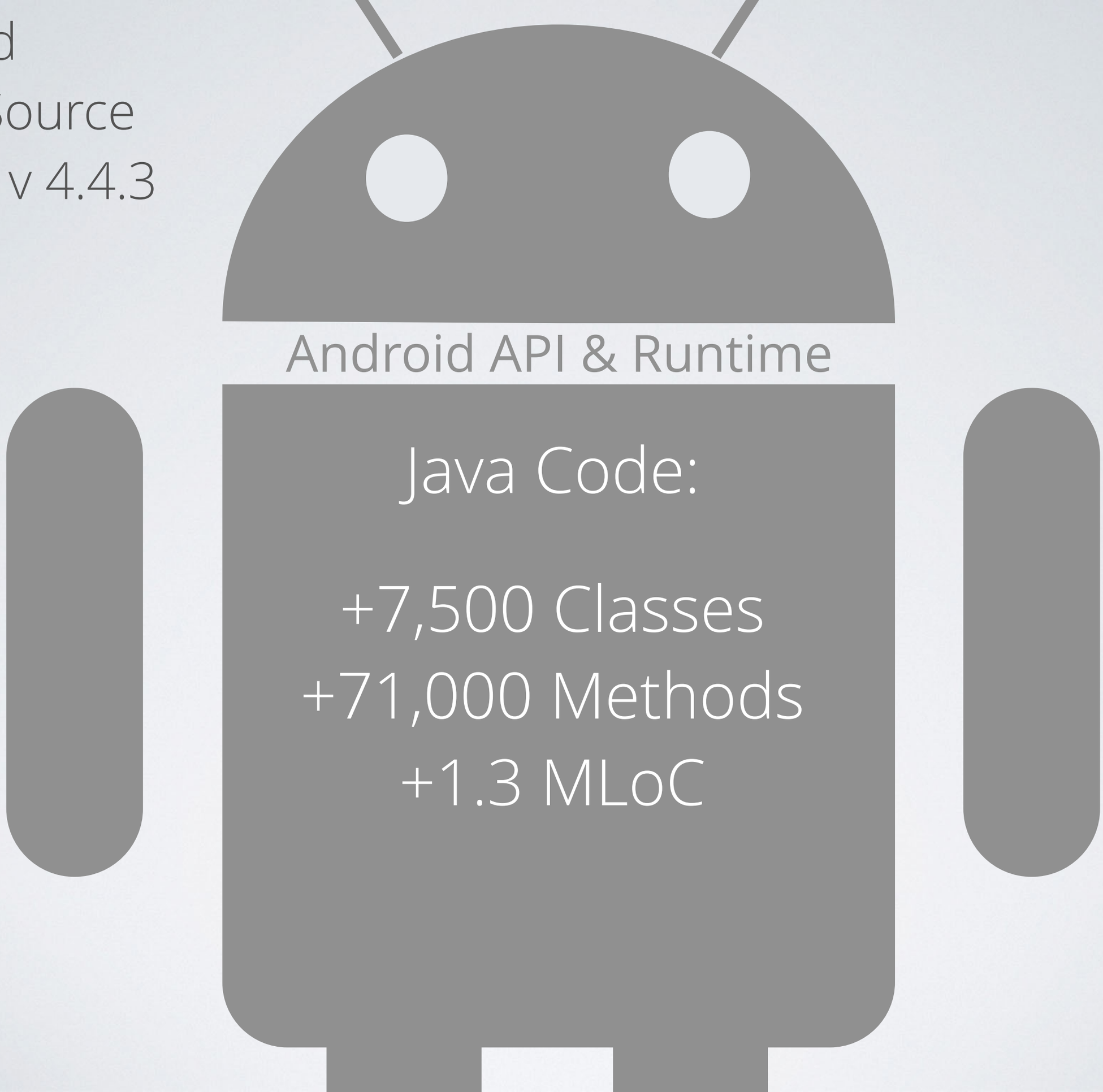
IMPLEMENTATION

ON DEMAND GLOBAL

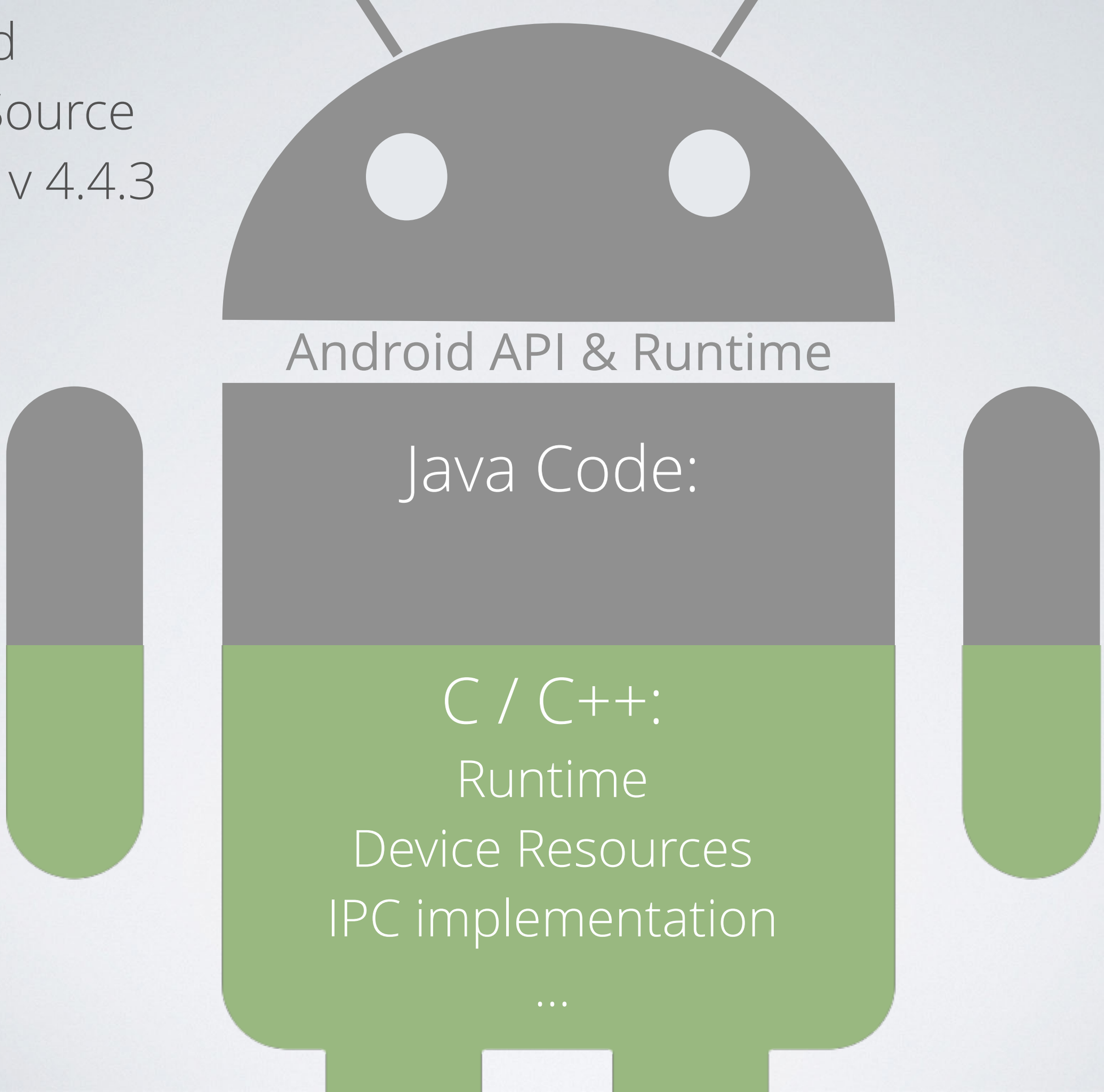
CUSTOM SOLVER GENERAL SOLVER

DroidSafe Model for the Android API and Runtime

Android
Open Source
Project v 4.4.3

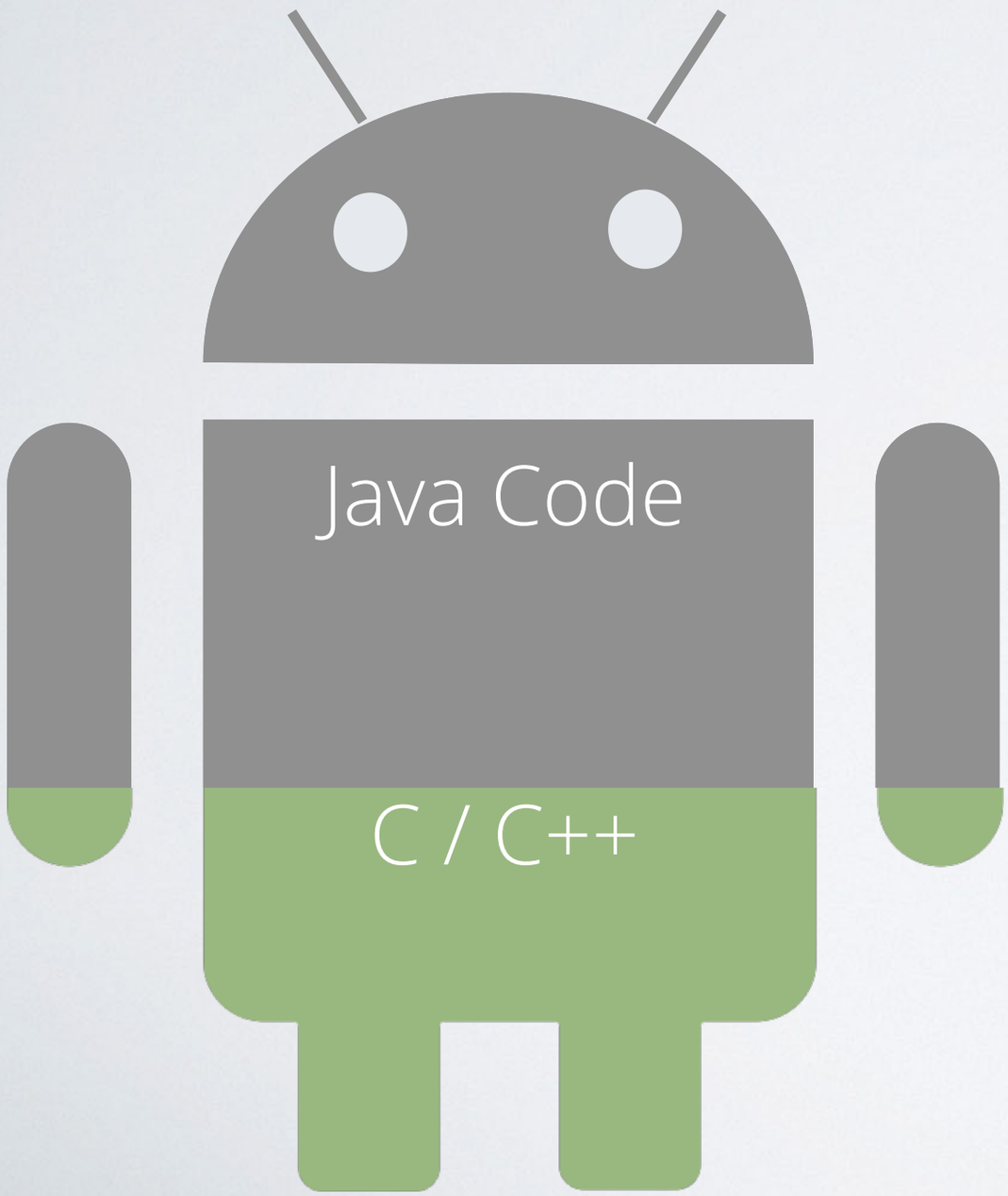


Android
Open Source
Project v 4.4.3



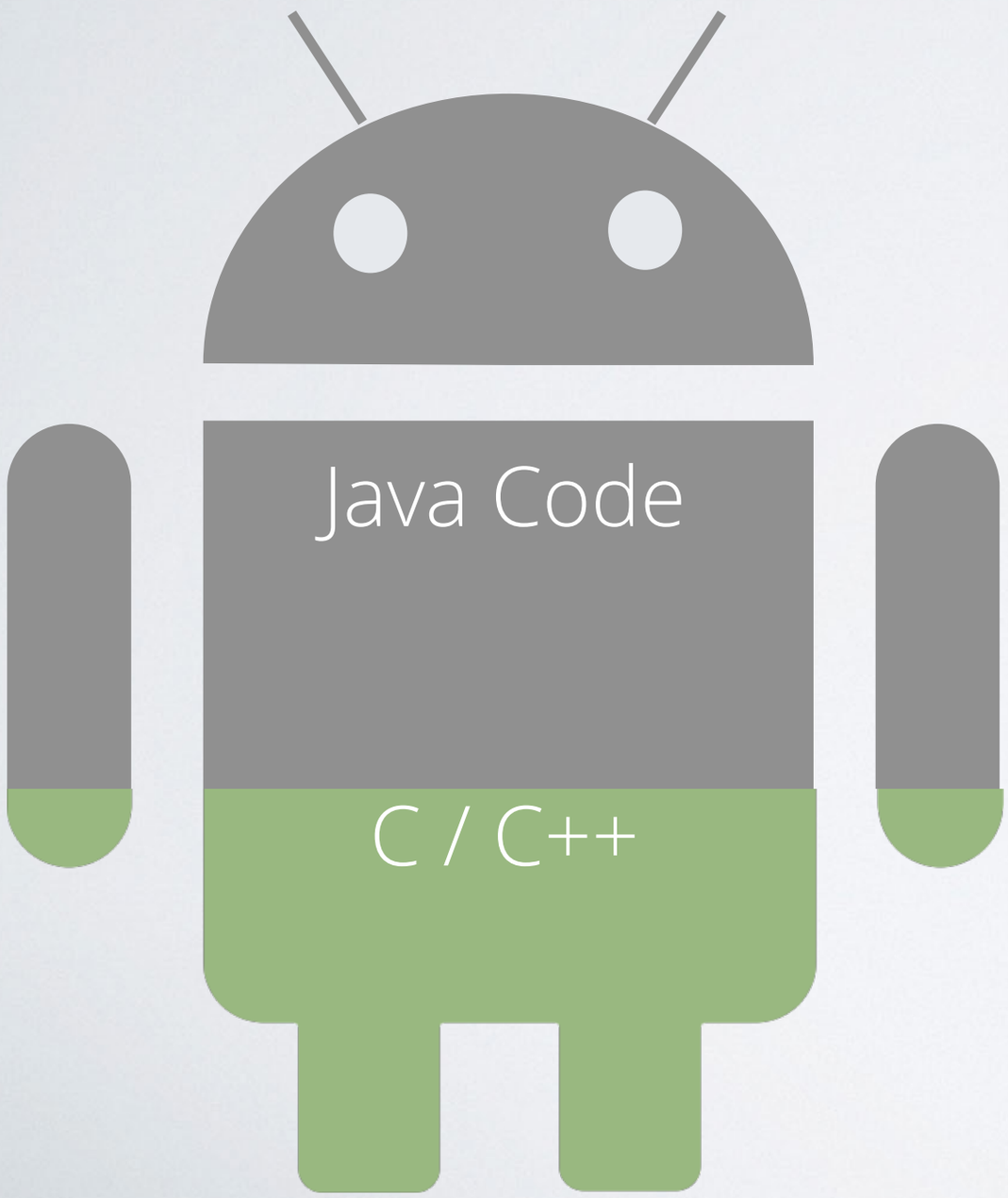
DroidSafe Model

AOSP
Implementation

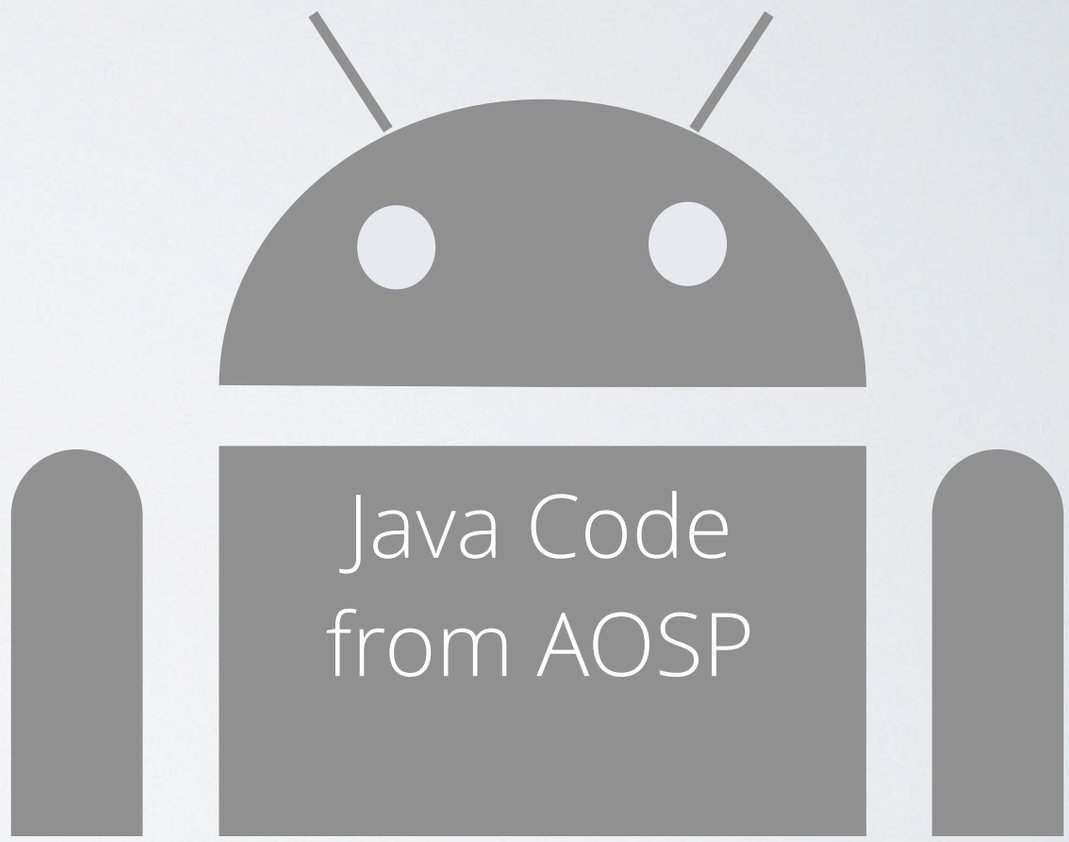


DroidSafe Model

AOSP
Implementation

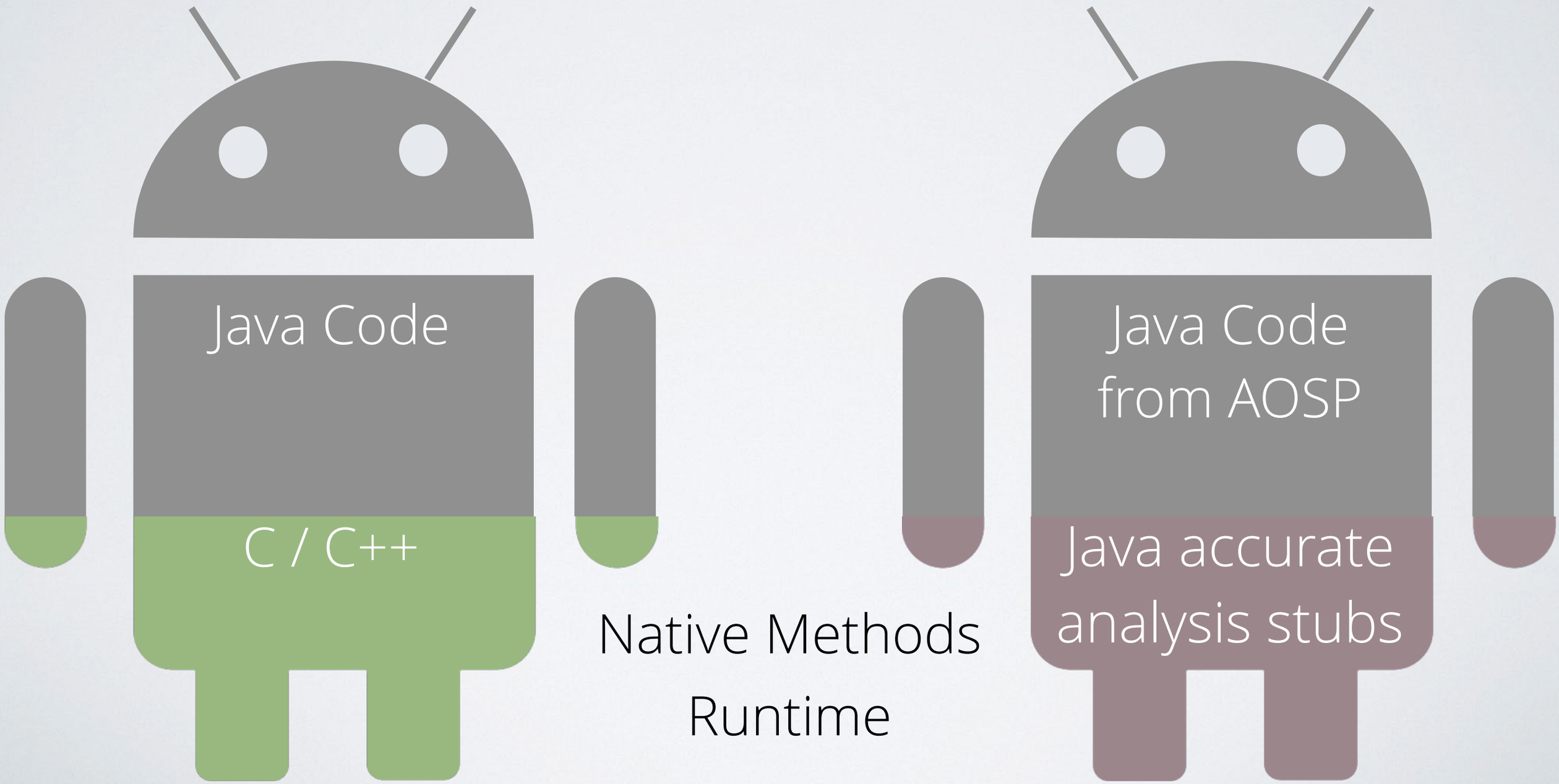


DroidSafe Model



AOSP
Implementation

DroidSafe Model



AOSP
Implementation

DroidSafe Model

Automated and manual process.

Details in paper.

Java Code

C / C++

Native Methods
Runtime

Java Code
from AOSP

Java accurate
analysis stubs

Java Accurate Analysis Stubs
Example: Parcel

AOSP Implementation

```
byte[] native_Marshall();
```

```
static jbyteArray android_os_Parcel_marshall(JNIEnv* env, jclass clazz, jint nativePtr)
{
    Parcel* parcel = reinterpret_cast<Parcel*>(nativePtr);
    if (parcel == NULL) {
       return NULL;
    }
    // do not marshall if there are binder objects in the parcel
    if (parcel->objectsCount())
    {
        jniThrowException(env, "java/lang/RuntimeException", “
Tried to marshall a Parcel that contained Binder objects.");
        return NULL;
    }
    jbyteArray ret = env->NewByteArray(parcel->dataSize());
    if (ret != NULL)
    {
        jbyte* array = (jbyte*)env->GetPrimitiveArrayCritical(ret, 0);
        if (array != NULL)
        {
            memcpy(array, parcel->data(), parcel->dataSize());
            env->ReleasePrimitiveArrayCritical(ret, array, 0);
        }
    }
    return ret;
}
```

Java Accurate Analysis Stubs
Example: Parcel

Java Accurate Analysis Stub

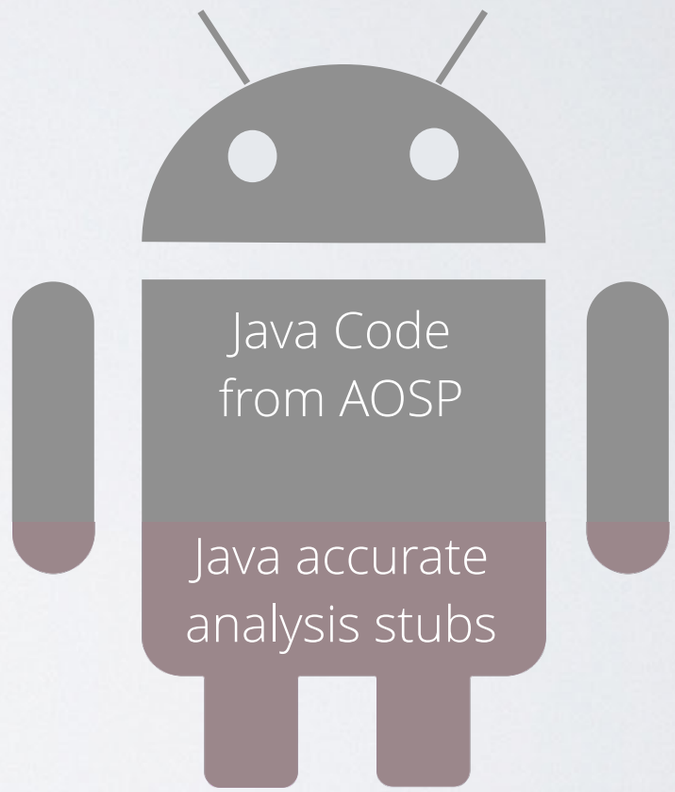
```
byte[] marshall() {
  byte[] ret = new byte[1];
  byte[0] = this.taint;
 return ret;
}

}
```

Android API & Runtime

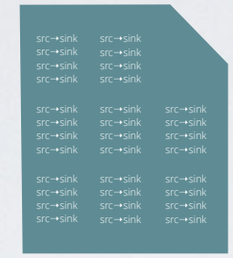
Not semantically
equivalent

$\neq$



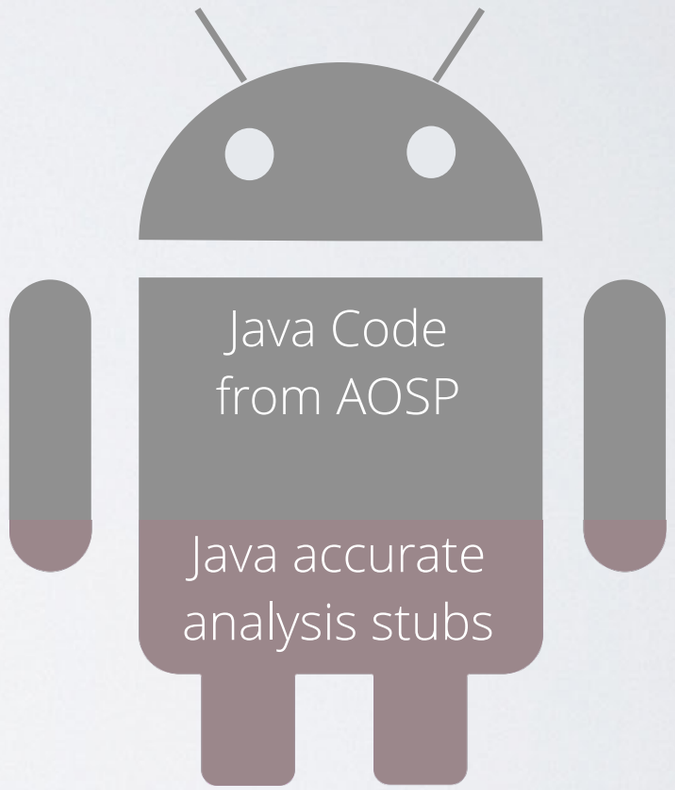
DroidSafe Model

Perfect Info Flow Analysis



Android API & Runtime

App

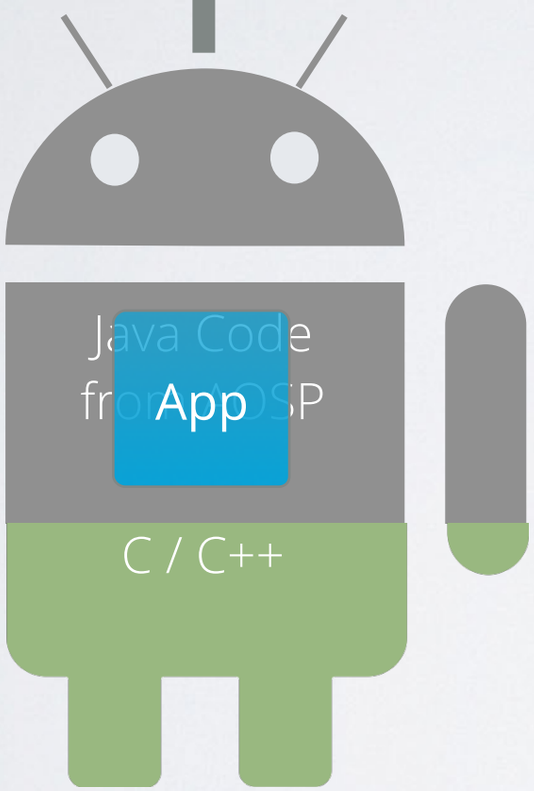


DroidSafe Model

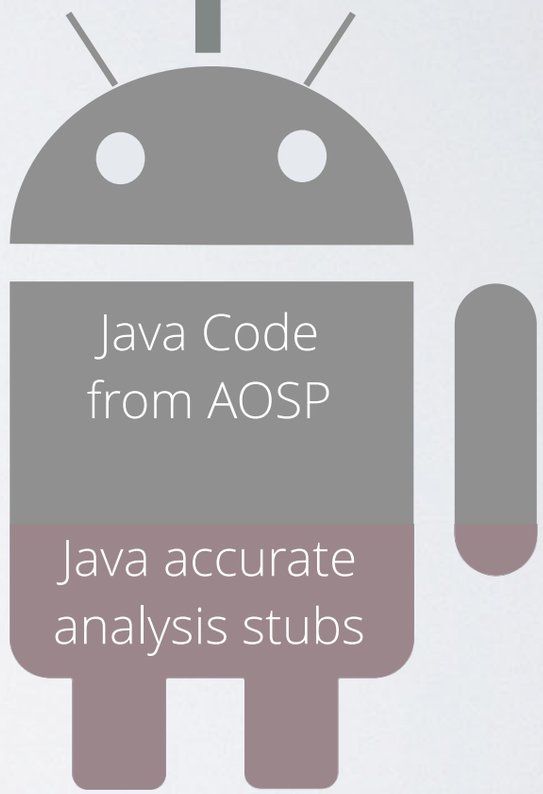
Perfect Info Flow Analysis

$\subseteq$

DroidSafe Analysis



Android API & Runtime



DroidSafe Model

DroidSafe Android Model: Android Device Implementation (ADI)

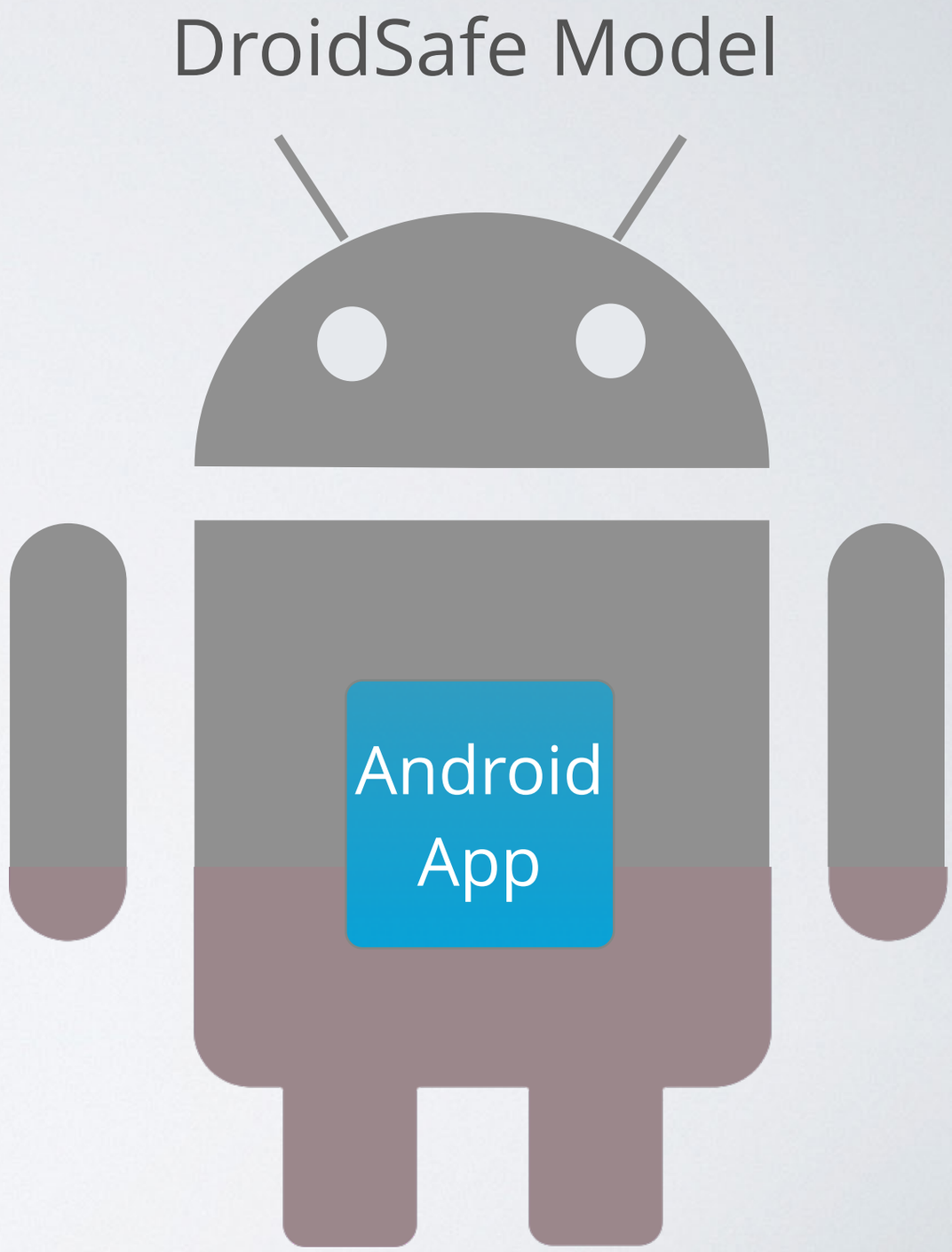
Comprehensive, accurate, and precise model of Android execution

- All semantics represented in Java
- Validated core that accounts for ~98% of calls in apps
- Component life-cycle event modeling
- Accurate and precise callback initiation and context

DroidSafe Static Analysis

Analysis in the Context of ADI

- On average, app reaches +200 KLoC in ADI
- Very difficult to achieve precision and scalability



Static Analysis Choices

Call-Site Context

Flow Sensitivity

ON  OFF

Field Sensitivity

ON  OFF

OBJECT SENSITIVITY

Heap Object Sensitivity

Method Object Sensitivity

IMPLEMENTATION

ON DEMAND  GLOBAL

CUSTOM SOLVER  GENERAL SOLVER

DroidSafe Static Analysis

Flow Sensitivity

ON  **OFF**

\+ Increased precision

\- Inadequate scalability for apps in context of Android model

\- Modeling event callback ordering error-prone

DroidSafe Static Analysis

Flow Sensitivity

ON  **OFF**

+ Adequate scalability for large apps in context of ADI

+ Relaxed requirements of callback modeling

Minor loss of precision compared to flow sensitivity

DroidSafe Static Analysis

Call-Site Context

Flow Sensitivity

ON  OFF

Field Sensitivity

ON  OFF

OBJECT SENSITIVITY

Heap Object Sensitivity

Method Object Sensitivity

IMPLEMENTATION

ON DEMAND  GLOBAL

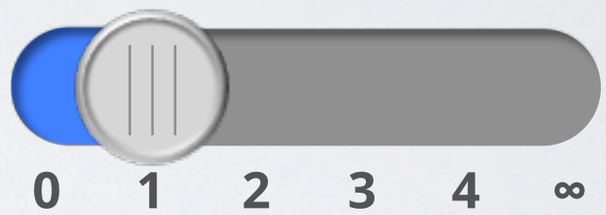
CUSTOM SOLVER  GENERAL SOLVER

DroidSafe Static Analysis

OBJECT SENSITIVITY

Heap Object Sensitivity

Method Object Sensitivity



Heavy reuse in our Android model means deep object-sensitivity required for precision

DroidSafe Static Analysis

OBJECT SENSITIVITY

Heap Object Sensitivity

Method Object Sensitivity

Deep object-sensitivity is expensive

DroidSafe Static Analysis

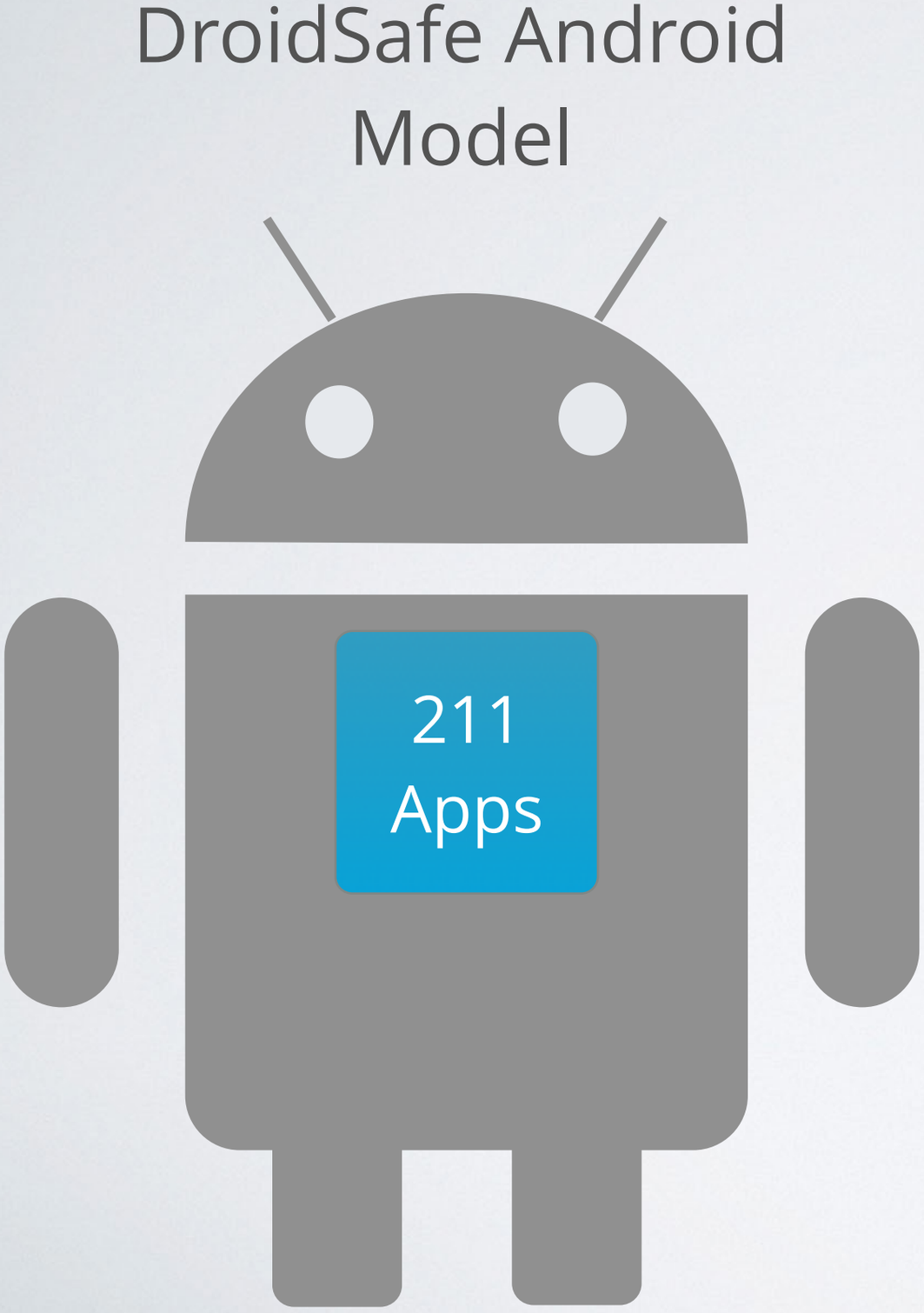
OBJECT SENSITIVITY

Heap Object Sensitivity

Method Object Sensitivity

For information flow analysis,
deep object sensitivity is not needed
for all classes of Android model.

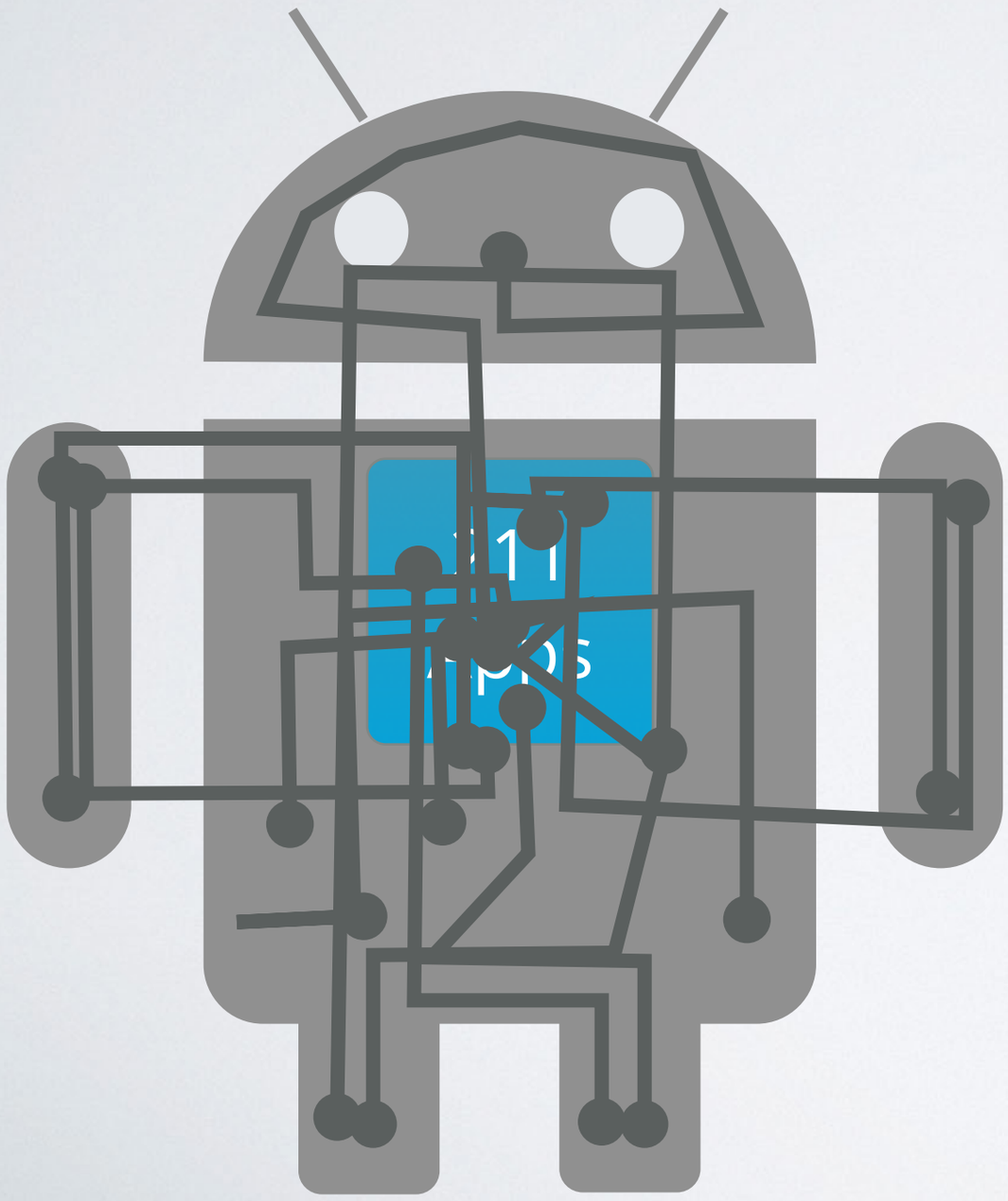
Selectively Applying Object Sensitivity



- We studied taint analysis results of 211 Android applications (both malicious and clean).

Selectively Applying Object Sensitivity

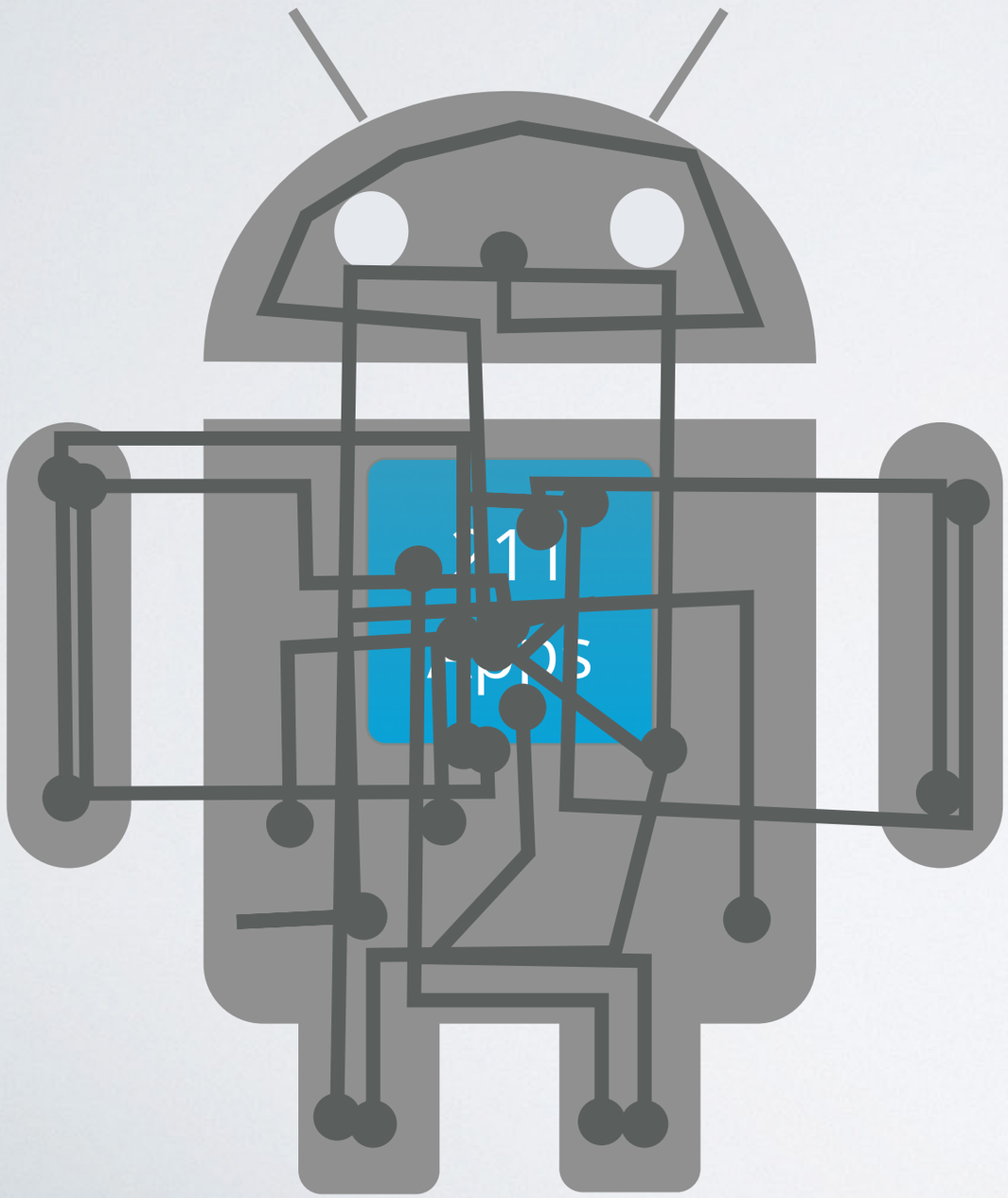
DroidSafe Android Model



- We studied taint analysis results of 211 Android applications (both malicious and clean).

Selectively Applying Object Sensitivity

DroidSafe Android Model



- We studied taint analysis results of 211 Android applications (both malicious and clean).
- Sensitive information does not flow through 26% of classes in our model.

Selectively Applying Object Sensitivity

- Analyze these 26% of classes with no context during analysis.
  - Still analyze the Java code
  - Still accurate if flows traverse these classes

- In practice, achieves near equivalent precision to uniform object-sensitivity.

- 5.1x analysis time savings over uniform object sensitivity.

DroidSafe Static Analysis

Call-Site Context

Flow Sensitivity

ON  OFF

Field Sensitivity

ON  OFF

OBJECT SENSITIVITY

Heap Object Sensitivity

Method Object Sensitivity

IMPLEMENTATION

ON DEMAND  GLOBAL

CUSTOM SOLVER  GENERAL SOLVER

Inter-component Communication

Inter-Component Communication

Android API & Runtime

Inter-Component Communication

Android API & Runtime

Communication mediated
by Android Runtime / API

Inter-Component Communication

Android API & Runtime

Targets identified by dynamic values such as Strings and object types.

Inter-Component Communication

Android API & Runtime

Taint analysis must consider these data flows.

Inter-Component Communication

Android API & Runtime

Precise targets when values can be resolved.

Conservative when values are unresolved.

DroidSafe ICC Modeling: Implementation Overview

- Run Java String Analyzer (JSA) [SAS 03] to calculate regular expressions for constructed String values.
- Model for `Intent` and `IntentFilter` built automatically from ADI classes.
- Global value analysis built on PTA to calculate model values
- Rewrite app intermediate representation patch data flow.
  - Framework for rapid development of support for ICC idioms

DroidSafe ICC Modeling

- The most complete, accurate, and precise model of Android ICC to date:
  - Starting and stopping `Service` and `Activity`
  - `Service` binding; send and receive `Service` messages; RPC on `Service`
  - `BroadcastReceiver` (including unregistered / dynamically created)
  - Dynamic `IntentFilter` registrations
  - `ContentProvider` operations

Evaluation

Methodology

- We compare to FlowDroid + IccTA [PLDI 2014]:
  - On demand, flow-sensitive, object-sensitive taint analysis
    - API summaries + blanket flow policies + simulated callback dispatch
    - IccTA adds inter-component communication resolution using EPICC [Usenix 2013]
- Use same source and sinks sets for FlowDroid and DroidSafe

Measurements

Accuracy
(Recall)

$$= \frac{\text{Reported True Flows}}{\text{Total True Flows}}$$

Precision

$$= \frac{\text{Reported True Flows}}{\text{Total Reported Flows}}$$

Experiment 1: Precision and Accuracy for Android Information Flow Benchmarks

DroidBench: A set of 94 applications developed by authors of FlowDroid and IccTA.

Experiment 1: DroidBench Results

	Accuracy	Precision
DroidSafe	93.9%	87.6%
FlowDroid + IccTA	80.6%	72.5%

DroidSafe reports 100% of explicit flows

Experiment 2: Does DroidSafe Capture Malicious Leaks in Sophisticated Malware?

- Set of 24 real-world APAC apps with malicious leaks of sensitive information
- Designed by independent, sophisticated red teams to stress analysis:
  - Flows through: ICC, Callbacks, complex Android idioms
- Aggressive malware for which malicious ground truth is established

APAC Application Size and Analysis Time

APAC Apps Size:

200 - 82,000 LoC

Avg: 10,000 LOC

DroidSafe Analysis Time:

Avg: 10 min

Max: 30 min

Experiment 2: Does DroidSafe Capture Malicious Leaks in Sophisticated Malware?

	Accuracy for Malicious Flows
DroidSafe	100%
FlowDroid + IccTA	9%

Experiment 2: Does DroidSafe Capture Malicious Leaks in Sophisticated Malware?

	Average Flows per App
DroidSafe	136
FlowDroid + IccTA	68

Conclusions

- Static analysis for Android requires a co-design of the Android runtime semantic model and analysis.
- DroidSafe provides a comprehensive, accurate, and precise model of Android runtime semantics.
- The DroidSafe static analysis achieves a balance between scalability and precision for this model.

DroidSafe is the only information flow analysis for Android applications that can provide acceptable accuracy and precision.